1st Edition

3.5 Git Branching - Externe Branches

Externe Branches

Externe (Remote) Branches sind Referenzen auf den Zustand der Branches in Deinen externen Repositorys. Es sind lokale Branches die Du nicht verändern kannst, sie werden automatisch verändert wann immer Du eine Netzwerkoperation durchführst. Externe Branches verhalten sich wie Lesezeichen, um Dich daran zu erinnern an welcher Position sich die Branches in Deinen externen Repositories befanden, als Du Dich zuletzt mit ihnen verbunden hattest.

Externe Branches besitzen die Schreibweise `(Repository)/(Branch)`. Wenn Du beispielsweise wissen möchtest wie der `master`-Branch in Deinem `origin`-Repository ausgesehen hat, als Du zuletzt Kontakt mit ihm hattest, dann würdest Du den `origin/master`-Branch überprüfen. Wenn Du mit einem Mitarbeiter an einer Fehlerbehebung gearbeitet hast, und dieser bereits einen `iss53`-Branch hochgeladen hat, besitzt Du möglicherweise Deinen eigenen lokalen `iss53`-Branch. Der Branch auf dem Server würde allerdings auf den Commit von `origin/iss53` zeigen.

Das kann ein wenig verwirrend sein, lass uns also ein Besipiel betrachten. Nehmen wir an Du hättest in Deinem Netzwerk einen Git-Server mit der Adresse `git.ourcompany.com`. Wenn Du von ihm klonst, nennt Git ihn automatisch `origin` für dich, lädt all seine Daten herunter, erstellt einen Zeiger an die Stelle wo sein `master`-Branch ist und benennt es lokal `origin/master`; und er ist unveränderbar für dich. Git gibt Dir auch einen eigenen `master`-Branch mit der gleichen Ausgangsposition wie origins `master`-Branch, damit Du einen Punkt für den Beginn Deiner Arbeiten hast (siehe Abbildung 3-22).

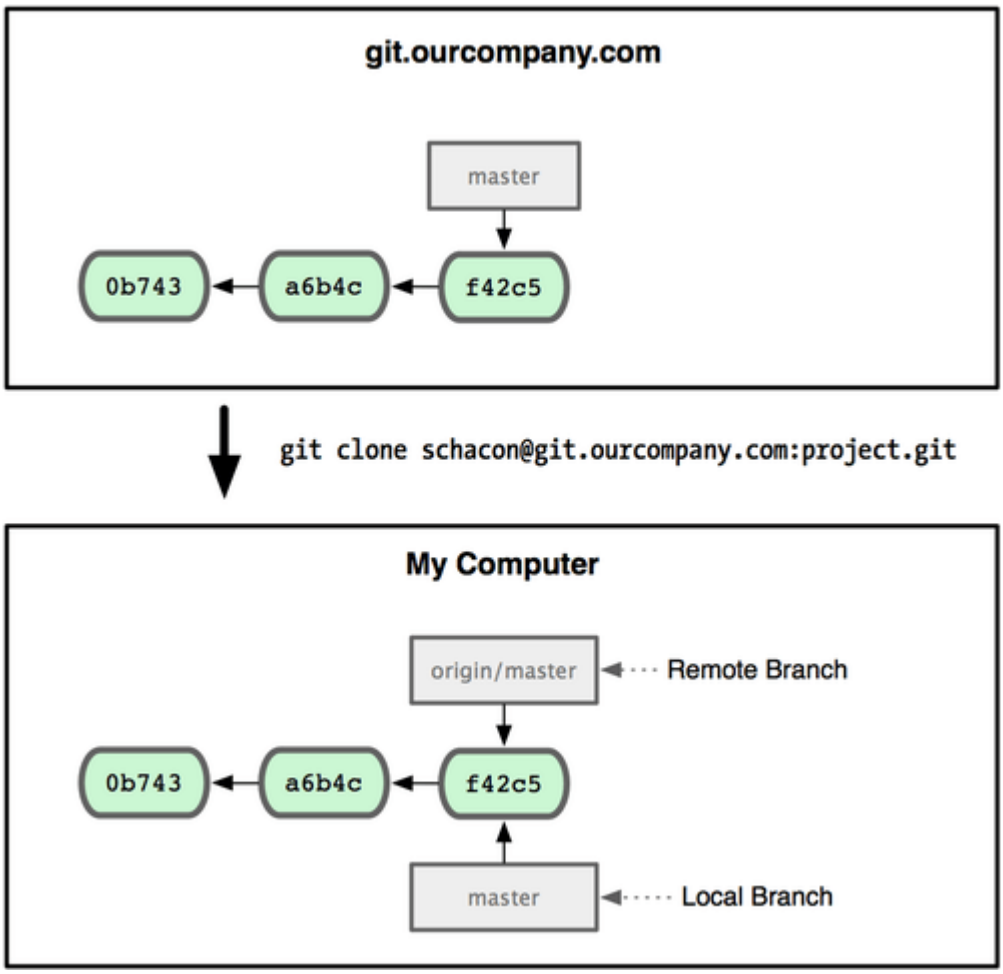

Abbildung 3-22. Ein 'git clone' gibt Dir Deinen eigenen `master`-Branch und `origin/master`, welcher auf origins 'master'-Branch zeigt.

Wenn Du ein wenig an Deinem lokalen `master`-Branch arbeitest und unterdessen jemand etwas zu `git.ourcompany.com` herauflädt, verändert er damit dessen `master`-Branch und eure Arbeitsverläufe entwickeln sich

unterschiedlich. Indes bewegt sich Dein `origin/master`-Zeiger nicht, solange Du keinen Kontakt mit Deinem `origin`-Server aufnimmst (siehe Abbildung 3-23).

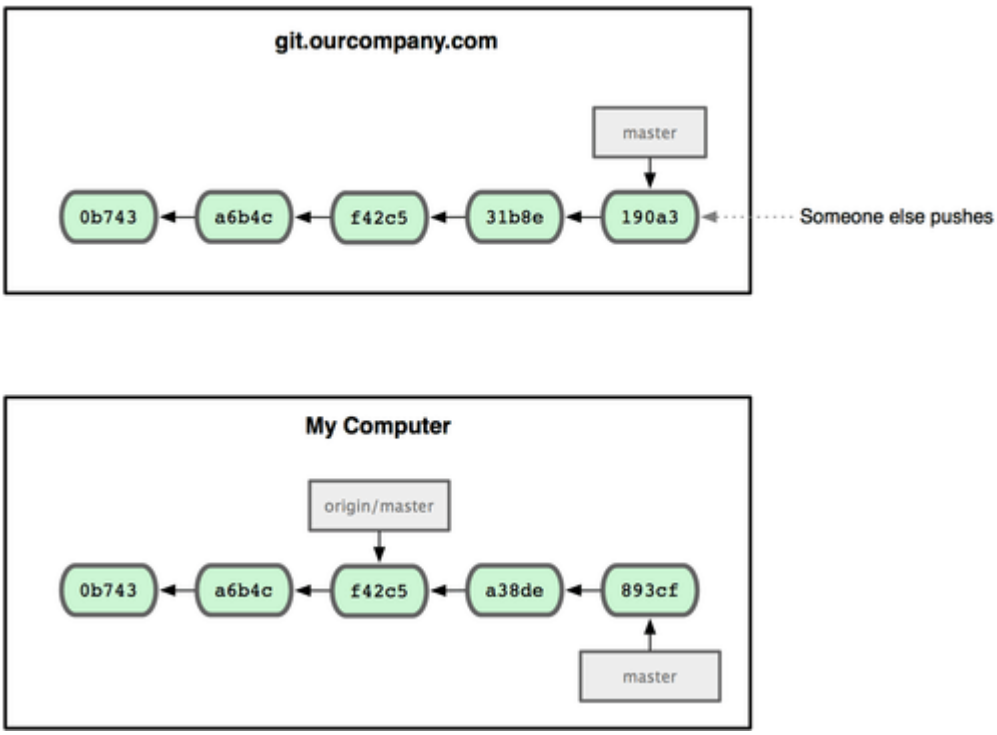

Abbildung 3-23. Lokales Arbeiten, während jemand auf Deinen externen Server hochlädt, lässt jeden Änderungsverlauf unterschiedlich weiterentwickeln.

Um Deine Arbeit abzugleichen, führe ein `git fetch origin`-Kommando aus. Das Kommando schlägt nach welcher Server `orgin` ist (in diesem Fall `git.ourcompany.com`), holt alle Daten die Dir bisher fehlen und aktualisiert Deine lokale Datenbank, indem es Deinen `orgin/master`-Zeiger auf seine neue aktuellere Position bewegt (siehe Abbildung 3-24).

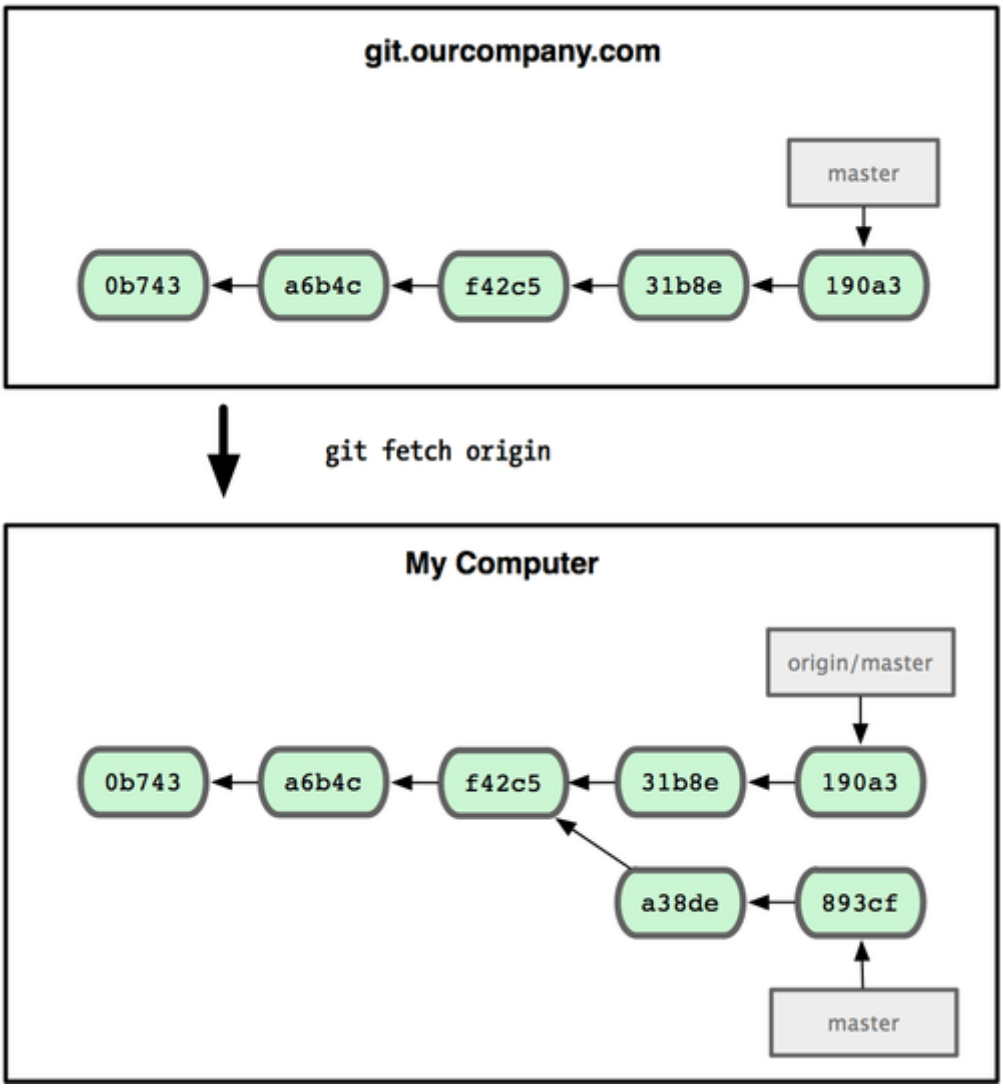

Abbildung 3-24. Das `git fetch`-Kommando aktualisiert Deine externen Referenzen.

Um zu demonstrieren wie Branches auf verschiedenen Remote-Servern aussehen, stellen wir uns vor, dass Du einen weiteren internen Git-Server besitzt, welcher nur von einem Deiner Sprint-Teams zur Entwicklung genutzt wird. Diesen Server erreichen wir unter `git.team1.ourcompany.com`. Du kannst ihn, mit dem Git-Kommando `git remote add` – wie in Kapitel 2 beschrieben, Deinem derzeitigen Arbeitsprojekt als weiteren Quell-Server hinzufügen. Gib dem Remote-Server den Namen `teamone`, welcher nun als Synonym für die ausgeschriebene Internetadresse dient (siehe Abbildung 3-25).

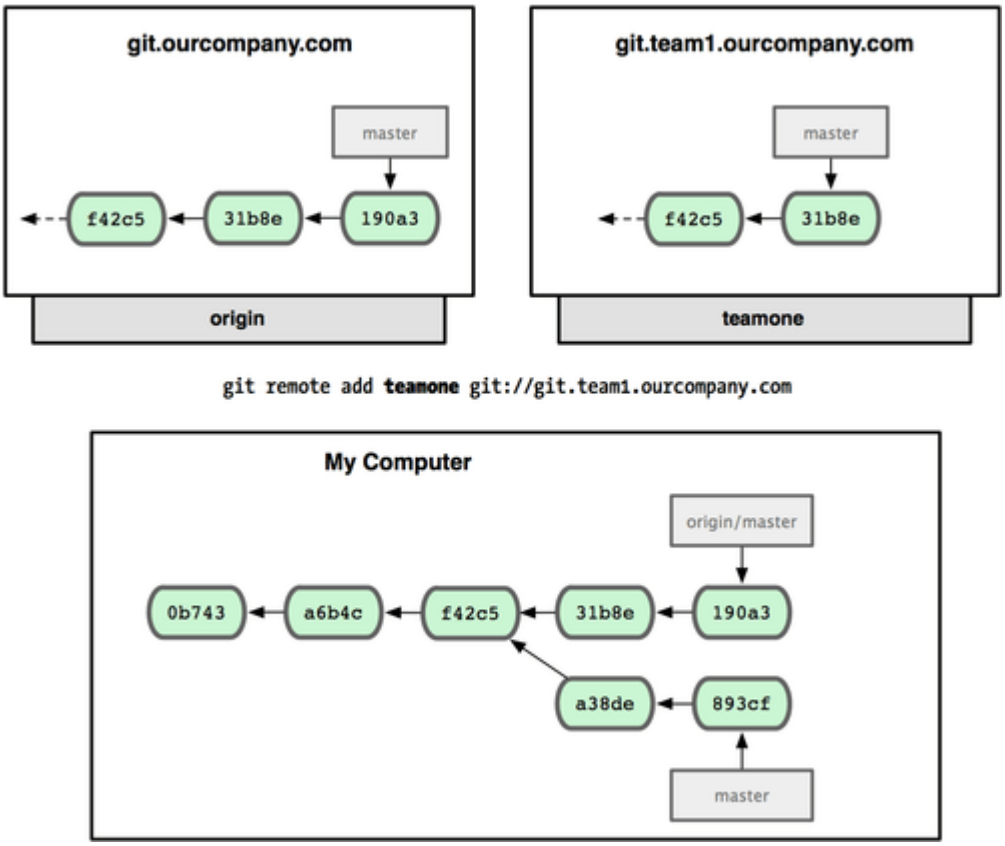

Abbildung 3-25. Einen weiteren Server als Quelle hinzufügen.

Nun kannst Du einfach `git fetch teamone` ausführen um alles vom Server zu holen was Du noch nicht hast. Da der Datenbestand auf dem Teamserver ein Teil der Informationen auf Deinem `origin`-Server ist, holt Git keine Daten, erstellt allerdings einen Remote-Branch namens `teamone/master`, der auf den gleichen Commit wie `teamone`s `master`-Branch zeigt (siehe Abbildung 3-26).

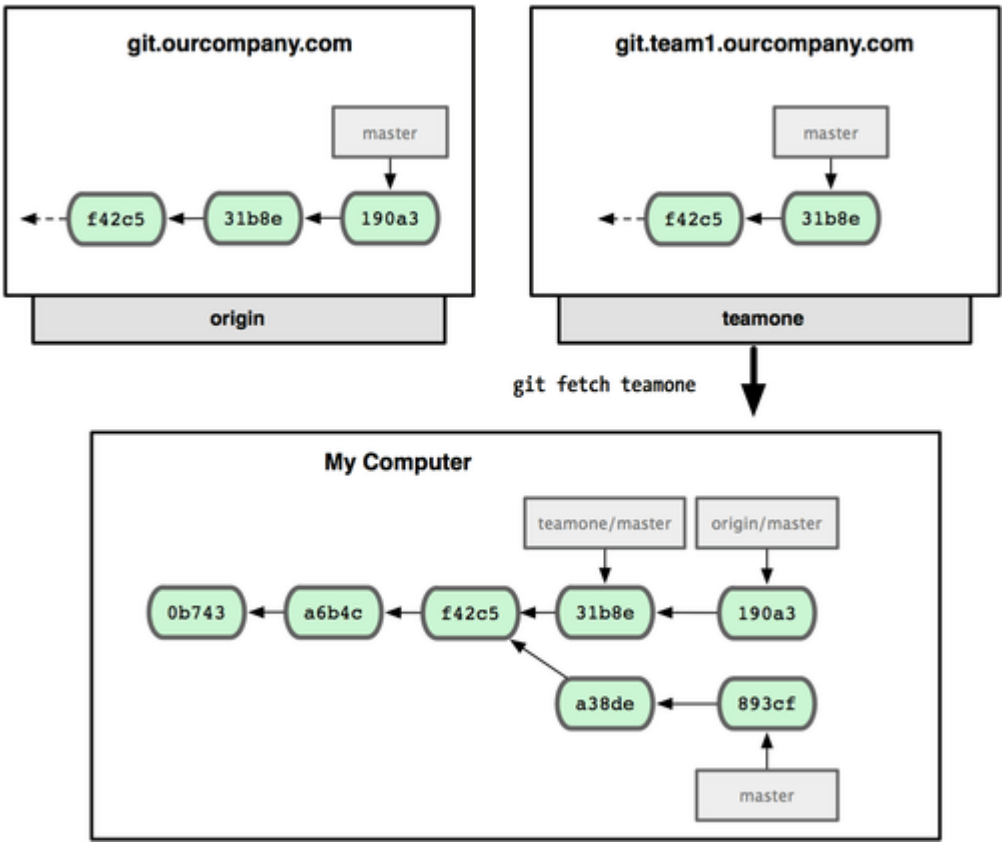

Abbildung 3-26. Du bekommst eine lokale Referenz auf `teamone`s `master`-Branch.

Hochladen

Wenn Du einen Branch mit der Welt teilen möchtest, musst Du ihn auf einen externen Server laden, auf dem Du Schreibrechte besitzt. Deine lokalen Zweige werden nicht automatisch mit den Remote-Servern synchronisiert wenn Du etwas änderst – Du musst die zu veröffentlichenden Branches explizit hochladen (pushen). Auf diesem Weg kannst Du an privaten Zweigen arbeiten die Du nicht veröffentlichen möchtest, und nur die Themen-Branches replizieren an denen Du gemeinsam mit anderen entwickeln möchtest.

Wenn Du einen Zweig namens `serverfix` besitzt, an dem Du mit anderen arbeiten möchtest, dann kannst Du diesen auf dieselbe Weise hochladen wie Deinen ersten Branch. Führe `git push (remote) (branch)` aus:

```
$ git push origin serverfix
Counting objects: 20, done.
Compressing objects: 100% (14/14), done.
Writing objects: 100% (15/15), 1.74 KiB, done.
Total 15 (delta 5), reused 0 (delta 0)
To git@github.com:schacon/simplegit.git
 * [new branch]      serverfix -> serverfix
```

Hierbei handelt es sich um eine Abkürzung. Git erweitert die `serverfix`-Branchbezeichnung automatisch zu `refs/heads/serverfix:refs/heads/serverfix`, was soviel bedeutet wie “Nimm meinen lokalen `serverfix`-Branch und aktualisiere

damit den `serverfix`-Branch auf meinem externen Server". Wir werden den `refs/heads/`-Teil in Kapitel 9 noch näher beleuchten, Du kannst ihn aber in der Regel weglassen. Du kannst auch `git push origin serverfix:serverfix` ausführen, was das gleiche bewirkt – es bedeutet "Nimm meinen `serverfix` und mach ihn zum externen `serverfix`". Du kannst dieses Format auch benutzen um einen lokalen Zweig in einen externen Branch mit anderem Namen zu laden. Wenn Du ihn auf dem externen Server nicht `serverfix` nennen möchtest, könntest Du stattdessen `git push origin serverfix:awesomebranch` ausführen um Deinen lokalen `serverfix`-Branch in den `awesomebranch`-Zweig in Deinem externen Projekt zu laden.

Das nächste Mal wenn einer Deiner Mitarbeiter den aktuellen Status des Git-Projektes vom Server abruft, wird er eine Referenz, auf den externen Branch `origin/serverfix`, unter dem Namen `serverfix` erhalten:

```
$ git fetch origin
remote: Counting objects: 20, done.
remote: Compressing objects: 100% (14/14), done.
remote: Total 15 (delta 5), reused 0 (delta 0)
Unpacking objects: 100% (15/15), done.
From git@github.com:schacon/simplegit
 * [new branch]      serverfix    -> origin/serverfix
```

Es ist wichtig festzuhalten, dass Du mit Abrufen eines neuen externen Branches nicht automatisch eine lokale, bearbeitbare Kopie derselben erhältst. Mit anderen Worten, in diesem Fall bekommst Du keinen neuen `serverfix`-Branch – sondern nur einen `origin/serverfix`-Zeiger den Du nicht verändern kannst.

Um diese referenzierte Arbeit mit Deinem derzeitigen Arbeitsbranch zusammenzuführen kannst Du `git merge origin/serverfix` ausführen. Wenn Du allerdings Deine eigene Arbeitskopie des `serverfix`-Branches erstellen möchtest, dann kannst Du diesen auf Grundlage des externen Zweiges erstellen:

```
$ git checkout -b serverfix origin/serverfix
Branch serverfix set up to track remote branch refs/remotes/origin/serverfix.
Switched to a new branch "serverfix"
```

Dies erstellt Dir einen lokalen bearbeitbaren Branch mit der Grundlage des `origin/serverfix`-Zweiges.

Tracking Branches

Das Auschecken eines lokalen Branches von einem Remote-Branch erzeugt automatisch einen sogenannten *Tracking-Branch*. Tracking Branches sind lokale Branches mit einer direkten Beziehung zu dem Remote-Zweig. Wenn Du Dich in einem Tracking-Branch befindest und `git push` eingibst, weiß Git automatisch zu welchem Server und Repository es pushen soll. Ebenso führt `git pull` in einem dieser Branches dazu, dass alle entfernten Referenzen gefetched und automatisch in den Zweig gemerged werden.

Wenn Du ein Repository klonst, wird automatisch ein `master`-Branch erzeugt,

welcher `origin/master` verfolgt. Deshalb können `git push` und `git pull` ohne weitere Argumente aufgerufen werden. Du kannst natürlich auch eigene Tracking-Branches erzeugen – welche die nicht Zweige auf `origin` und dessen `master`-Branch verfolgen. Der einfachste Fall ist das bereits gesehene Beispiel in welchem Du `git checkout -b [branch] [remotename]/[branch]` ausführst. Mit der Git-Version 1.6.2 oder später kannst Du auch die `--track`-Kurzvariante nutzen:

```
$ git checkout --track origin/serverfix
Branch serverfix set up to track remote branch serverfix from origin.
Switched to a new branch 'serverfix'
```

Um einen lokalen Branch mit einem anderen Namen zu versehen als jenem im Remote-Branch, kannst Du einfach die erste Variante mit einem neuen lokalen Branch-Namen verwenden:

```
$ git checkout -b sf origin/serverfix
Branch sf set up to track remote branch serverfix from origin.
Switched to a new branch 'sf'
```

Nun wird Dein lokaler Branch `sf` automatisch push und pull auf `origin/serverfix` durchführen.

Löschen entfernter Branches

Stellen wir uns vor, Du bist fertig mit Deinem Remote-Branch – sagen wir Deine Mitarbeiter und Du, Ihr seid fertig mit einer neuen Funktion und habt sie in den entfernten `master`-Branch (oder in welchem Zweig Ihr sonst den stabilen Code ablegt) gemerged. Du kannst einen Remote-Branch mit der unlogischen Syntax `git push [remotename] :[branch]` löschen. Wenn Du Deinen `serverfix`-Branch vom Server löschen möchtest, führe folgendes aus:

```
$ git push origin :serverfix
To git@github.com:schacon/simplegit.git
 - [deleted]         serverfix
```

Boom. Kein Zweig mehr auf Deinem Server. Du möchtest Dir diese Seite vielleicht markieren, weil Du dieses Kommando noch benötigen wirst und man leicht dessen Syntax vergisst. Ein Weg sich an dieses Kommando zu erinnern führt über die `git push [remotename] [localbranch]:[remotebranch]`-Snytax, welche wir bereits behandelt haben. Wenn Du den `[localbranch]`-Teil weglässt, dann sagst Du einfach „Nimm nichts von meiner Seite und mach es zu `[remotebranch]`“.

prev | next